Recall: For Loops

- loop sequence: grades
- loop variable: x
- body: print(x)

To execute the for-loop:
1. Check if there is a “next” element of loop sequence
2. If so:
   - assign next sequence element to loop variable
   - Execute all of the body
   - Go back to Line 1
3. If not, terminate execution:

---

Different types of Repetition

1. Process each item in a sequence
   - Compute statistics for a dataset
   - Send all your contacts an email
2. Do something n times
   - Draw a checkers board
   - Run a protein-folding simulation for 10^6 time steps
3. Do something an unknown number of times
   - Play word guessing game until 6 strikes
   - Go in current direction until edge is detected

---

While-Loops and Flow

```python
import random
num = random.randint(0,10)
guessed_it = False
print("I'm thinking of a number.")
while not guessed_it:
    guess = int(input('Guess it: '))
guessed_it = (num == guess)
print('Well done!')
```

Q1: What gets printed?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 0</td>
<td>a = 0</td>
<td>a = 0</td>
</tr>
<tr>
<td>while a &lt; 1:</td>
<td>while a &lt; 2:</td>
<td>while a &gt; 2:</td>
</tr>
<tr>
<td>a = a + 1</td>
<td>a = a + 1</td>
<td>a = a + 1</td>
</tr>
<tr>
<td>print(a)</td>
<td>print(a)</td>
<td>print(a)</td>
</tr>
</tbody>
</table>
Q2: What gets printed?

```python
a = 4
while a > 0:
    a = a - 1
print(a)
```

```python
a = 0
while a < 3:
    if a < 2:
        a = a + 1
    print(a)
```

Q3: What gets printed?

```python
a = 8
b = 12
while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a
print(a)
```

This is Euclid's Algorithm for finding the greatest common factor of two positive integers. **Trivia:** It is one of the oldest recorded algorithms (~300 B.C.)

A: Infinite loop
B: 8
C: 12
D: 4
E: I don't know

---

for vs. while

- You can almost always use either
  - Sometimes for is better
    - Do something a fixed (pre-determined) number of times
  - Sometimes while is better
    - Do something an indefinite (not infinite) number of times
    - E.g., do something until some event happens, i.e., until a stopping condition is reached

for vs. while

- do something n times
  ```python
  for k in range(n):
      # do something
  ```

- do something an unknown number of times
  ```python
  for k in range(BIG_NUM):
      # do something
  ```
  ```python
  while not time to stop:
      # do something
      if time to stop:
          break
  ```

My preference? for-loop
My preference? while-loop

Do NOT use break in any work you submit in CS1110. Practice using while-loop in situations where while-loop is well suited.
for vs. while

do something to each element of a sequence

```python
for k in range(len(seq)):
    seq[k] = seq[k]+1
```

```python
k = 0
while k < len(seq):
    seq[k] = seq[k]+1
    k = k+1
```

while is more flexible, but sometimes requires more code

My preference? for-loop

for vs. while

do something until a limit is reached
e.g., make a table of squares up to N

```python
seq = []
for k in range(sqn+1):
    seq.append(k*k)
```

```python
seq = []
for k in range(len(seq)):
    seq[k] = seq[k]+1
```

for-loop requires you to know how many iterations you want 
ahead of time

My preference? while-loop

for vs. while

change a sequence's length
e.g., remove all 3's for list nums

```python
for i in range(len(nums)):
    if nums[i] == 3:
        del nums[i]
```

```python
while 3 in nums:
    nums.remove(3)
```

IndexError: list index out of range

is this not beautiful?

My preference? while-loop

for vs. while

change a sequence's length
e.g., remove all 3's for list nums

```python
Fibonacci numbers:
F_0 = 1
F_1 = 1
F_n = F_{n-1} + F_{n-2}
```

```python
fib = [1, 1]
for k in range(2,n):
    fib.append(fib[-1] + fib[-2])
```

```python
fib = [1, 1]
while len(fib) < n:
    fib.append(fib[-1] + fib[-2])
```

loop variable not always used

loop variable not always needed at all

My preference? while-loop

Using while-loops Instead of for-loops

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Better for modifying data</td>
<td>• Infinite loops more likely</td>
</tr>
<tr>
<td>• More natural than range</td>
<td>• Easy to forget loop vars</td>
</tr>
<tr>
<td>• Works better with deletion</td>
<td>• Or get continuation condition wrong</td>
</tr>
<tr>
<td>• Better for convergent tasks</td>
<td>• Require more management</td>
</tr>
<tr>
<td>• Loop until calculation done</td>
<td>• Initialize the condition?</td>
</tr>
<tr>
<td>• Exact steps are unknown</td>
<td>• Update the condition?</td>
</tr>
<tr>
<td>• Easier to stop early</td>
<td></td>
</tr>
<tr>
<td>• Just set loop var (keep_growing) to False</td>
<td></td>
</tr>
</tbody>
</table>

Start next video:
How to set up a while loop
Setting up a while-loop

0. Situation is to do something until an event happens
1. Write the continuation condition
   * Create var names as necessary to express condition
   * May be easier to negate stop condition to get continuation condition
2. Initialize loop vars (vars in loop condition) as necessary
3. In loop body: update loop vars
to possibly change loop condition from True to False
4. Write the rest of the loop body

Improve number guessing game

```python
import random
min_num= 1
max_mum= 10
max_chances= 5
secret_num= random.randint(min_num, max_mum)
print("I have a number from "+str(min_num)+" to "+str(max_mum))
print("You have "+str(max_chances)+" chances to guess it")
# User guesses until all chances used up or guessed correctly
```
1. Allow fixed number of guesses
   For you to add later:
   2. If a guess is wrong, tell player whether it was too high or too low.